AFIT/EN-TR-92-4

Air Force Institute of Technology

An Eclectic Method for Object-Oriented Database Design

Douglas E. Dyer
Capt, USAF

Mark A. Roth
Maj, USAF

9 September 1992

An Eclectic Method for Object-Oriented Database Design*

Douglas E. Dyer and Mark A. Roth

Department of Electrical and Computer Engineering (AFIT/ENG)
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433-6583

Abstract

No dominant methodology has emerged for designing object-oriented databases. In this paper, we identify characteristics for a good design methodology and review the literature of supporting models and approaches. Semantic data models and object-oriented methods are presented. We then take an extended look at Harel's higraph notation and use it to extend the entity-relationship model for object-oriented database design.

1 Introduction

The entity-relationship (E-R) data model proposed by Chen is widely used to design relational databases. The E-R model is successful because (1) it captures the essential concepts of the real-world, (2) it clearly communicates these concepts in the form of entity-relationship diagrams, (3) it is generally useful for many modeling applications, including, hierarchical, network, and relational data models, and (4) it has been extended to cover its previous shortcomings.

New databases built under object-oriented database management systems cannot be modeled adequately with the E-R semantic model. The primary shortcoming is the inability to model the procedural abilities of active data objects, i.e., behaviors. The E-R diagram is also somewhat inadequate at modeling data which are hierarchically related.

In this paper, we first establish some goals for an object-oriented database (OODB) design methodology. After that, we review the literature for design approaches and models which could be applied to designing databases which operate under an object data management system (ODMS).[1] There are a number of OODB design approaches by various researchers, some of which extend the E-R model and others which explore different semantic models. There are also applicable design approaches from the object-oriented programming community. Some of these object-oriented approaches have been tailored for OODB design. Two other models which might influence OODB design are also presented.

Several researchers have begun to use Harel's higraph notation for modeling dynamics. It turns out that this same notation is also useful for improving E-R diagrams. We take an extended look at higraphs as E-R diagrams and Statecharts as tools for modeling structural and behavioral characteristics of data. Finally, we present an OODB design methodology based on ideas from the literature and this unified higraph notation.

2 Goals for an OODB Design Methodology

Designing any complex system correctly depends on the use of a disciplined methodology. A design methodology is a general process which can be tailored to suit the needs of the designer. Invariably, a successful design methodology is an iterative procedure in which abstract designs become more concrete, unworkable designs are fixed, and omissions or errors are corrected. Because of the iterative nature of design, the design methodology should be characterized by a series of activities and products. The design activities involve matching the user requirements with ODMS capabilities

*Research funded by ASC/RWWW, Wright-Patterson AFB, OH

[1] This term appears in [6]. We are using it to refer to database management systems based on extended object-oriented programming languages, although most comments are also applicable to extended relational systems. For the most part, we will use terminology and assumptions made in [6] throughout this paper. Because of the state of flux in ODMSs, these assumptions are somewhat mandatory.

to synthesize a design which meets all needs. In general, design products are diagrams and text specifications which represent design choices made.

Expressive design diagrams and notation are a critical tool for any design methodology for two reasons. First, as the product of a design phase, diagrams serve as a communication bridge between different developers and between developers and customers. Second, diagrams aid in the design process itself by capturing design ideas and freeing the designer to think about other parts of the problem. With a diagram, the designer doesn't have to remember the details of what has already been designed. Diagrams shape the mind set of the designer, and thus alter the design in subtle ways. Because diagrams and notation are so important, this paper focuses closely on diagrams as design tools.

Diagrams and notation should be able to naturally represent all aspects of the real-world at different abstraction levels. However, diagram semantics should be well-understood by both designers and users. These two requirements are not independent; diagrams should be semantically rich, yet simple. Furthermore, these design products should be easy to transform into a the set class and object definitions, together with a specification for any object procedures required. Finally, it should be easy to detect errors while implementing the design [15, 23]. Metaphorically, the design products should be rich and robust enough to serve as both a requirements contract and a design blueprint.

Once diagrams and notations have been chosen, a design process, or set of design activities, is needed. In general, design is not well understood. A good design methodology can suggest a series of design activities, but it is up to the designer to know how to apply them, when to iterate, and where to focus attention. A good design methodology cannot guarantee a good design. Instead, it only serves as a guide.

For OODB design, we need to be able to identify classes and objects in the domain of interest. We need to understand the semantics of each class and object, including what attributes are important and what type and domain constraints exist. We need to know how classes and objects relate to one another, and what constraints exist on relationships. In addition to understanding these structural aspects, we must have a grasp for the intra- and inter-object behaviors exhibited. We need to be able to state the effects of these behaviors in a declarative way. Finally, we need to be able to model, within the framework of our ODMS, all of these things and be able to show that the model adequately covers all needs.

3 A Survey of Models Useful for Object-Oriented Design

Database design methodologies and diagrams are normally based on some type of abstract data model. While there are many different approaches, we can broadly divide them into two groups. Semantic data models like the E-R model form the basis for many design methodologies. Other design methods are based on the object-oriented software design methods.

In this section, we review the semantic and object-oriented models which have been proposed for OODB design. The presentation is necessarily broad; the goal of the discussion is to qualify the contribution of the work and the suitability of the model for OODB design.

3.1 Semantic Models

3.1.1 The Entity-Relationship Model and Extensions

The entity-relationship model proposed in [7] is well-known to database designers. Entities are distinguishable real-world objects characterized by attributes and relationships between other entities. Extensions to Chen's original model include concepts of generalization and aggregation which appear in [26]. The E-R model has a corresponding E-R diagram, such as the one shown in Figure 1, which should be familiar to all readers. An alternative notation, one which uses a different symbol for the special `IS-A` relationship, is shown in Figure 2. These figures capture the model of a software development firm. For simplicity, neither attributes nor cardinality constraints are shown in these figures, although they are common features of E-R diagrams. During design, the E-R diagram is tightly coupled to the use of the E-R model and serves as a communication medium between users and database implementors.

The E-R model represents most integrity constraints adequately, but some real-world semantic constraints and all general procedures are handled ad hoc. Cardinality and key constraints appear on the E-R diagram. Domain constraints can also appear on the diagram, but are often suppressed. Referential integrity constraints are maintained when the E-R model is transformed into a relational database. Also, transformation to relational database tables typically results in a database in Boyce-Codd or third normal form [20].

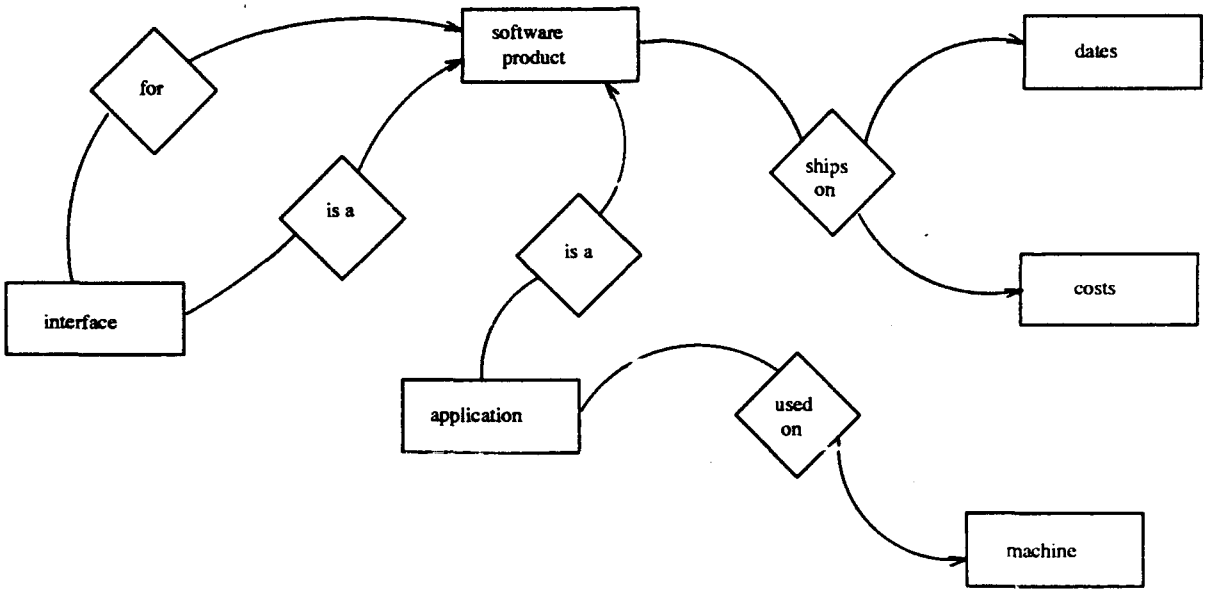


Figure 1: An entity-relationship diagram

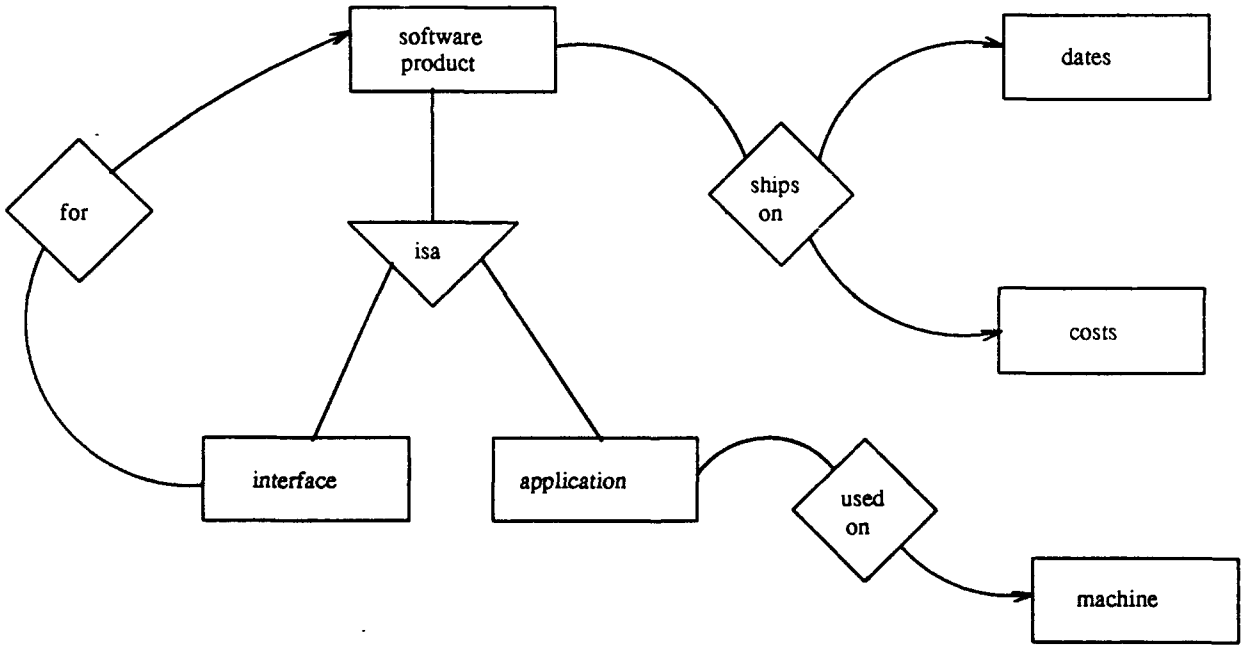


Figure 2: An alternative E-R notation for IS-A

Unfortunately, the E-R model is not as appropriate for OODB design as it is for the development of relational databases because it cannot specify object behaviors. For example, the E-R model fails to support additional constraints which may arise from the real-world. In practice, we might want to assert that no software product should have a *cost* less than $50. As another example, we may have a dynamic integrity constraint which says that entered product data may not be deleted. ODMS's, at least those based on an object-oriented programming language, can maintain these constraints. Unfortunately, the E-R diagram can't show these types of constraints; if the E-R model is used for design, constraint information must be included in a text write-up. Furthermore, other types of more general procedural calls which are also possible in an ODMS cannot be represented in the E-R diagram. An example of a procedural call is a trigger, which is a rule that fires as a side effect of database modification.

Because of the utility and popularity of the E-R model, some researchers are beginning to extend the E-R model for use in OODB design. For example, Navathe and Pillalamarri [22] define abstractions of key ideas in the E-R and other semantic models. These key ideas are aggregation, generalization, classification, association, and identification. Abstractions for each of these ideas are defined in terms of structure, constraints, operations, and behavior. *Aggregation* is the abstraction used to create entities or objects. Typically, aggregates are the `A-PART-OF` relationships. *Generalization* refers to the `IS-A` relationships supported by the E-R model. *Classification* is based on the notion of a set and serves mainly to differentiate between individual instances and a set or class of those instances. The idea of *association* abstracts relationships between objects. *Identification* is unique to ODMS's, where objects are identified uniquely and have a persistent identity which is independent of state change. An interaction matrix among the abstractions is presented in [22], as well as diagramatic nomenclature.

The contribution of [22] is to unify ideas in the E-R model with those of other semantic models which have been proposed for for OODB design. Unfortunately, the model developed does not address behavioral capabilities of a typical ODMS. Moreover, the structure graphs presented depend heavily on shapes, rather than topology, which would have more cleanly communicated many of the abstractions developed.

While [22] did not integrate behavioral capabilities of object-oriented data models into the E-R model, that is the focus of [18], which presents a model called the behavior integrated entity-relationship (BIER) approach. In [18], Kappel and Schreff state:

> The *design goal* is to have the behavior of an object defined solely by its process. Then the activities of the process characterize the set of operations which may be applied on the object. As a consequence, the description of the object in the static model can be encapsulated with the description of the activities in the behavior model to an object class of an object-oriented database.
>
> The *design problem* is to reflect the effects of the real world events correctly in the process types of the individual objects which participate in the event.

The approach used in BIER is to model structure using E-R diagrams and model object dynamics using Petri nets. Petri nets are often used to model simulation and queuing problems. A Petri net consists of a set of states and a set of legal transitions between states. Transitions can have multiple pre-states and multiple post-states. States are characterized by owning tokens. If all pre-states of a transition have a token, the transition is fired and a token is removed from each pre-state, while each post-state is assigned a token. As an example, in Figures 3 and 4, there is a model of software development and distribution. Figure 3 shows a linear software development process, beginning with specification and ending with shipping the product. When a token is available in some state prior to $a1$, the Petri net rules specify the depicted order of transitions which terminate in the SOFTWARE SHIPPED state.

Complex activities can be modeled by BIER methods. For example, Figure 4 shows how the distribution of software can be comprised from elementary activities for *ship-software* and *accept-distribution*. Complex activities are aggregates of elementary activities of lower-level objects so that the effects of the complex activities are reflected in participating objects.

Translation of the BIER method into a OODB schema is presented in [18]. Class data structures and variables are obtained from the structural, E-R portion, while methods arise from the Petri net part of the BIER model.

The BIER model addresses many needs of an OODB design method. In particular, the BIER approach can communicate partial orderings on activities quite clearly because of the choice of Petri nets as a dynamic modeling medium. While the BIER model fails to specifically address constraints and exceptions, the Petri net notation used can be modified for these purposes. However, Petri nets are expressively equivalent to state diagrams, which are more commonly used to

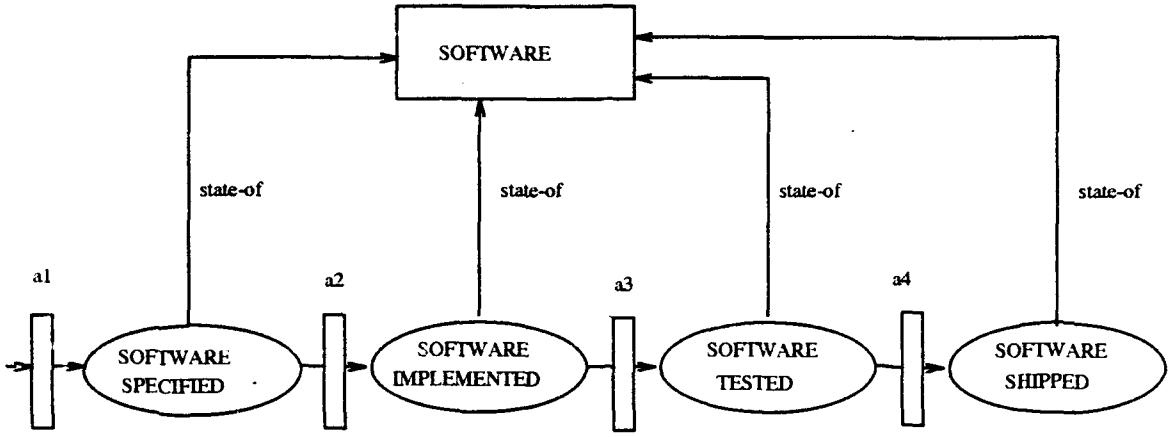


Figure 3: Software Development Process

model system dynamics and object behaviors. A later paper by the same authors extend the ideas in the BIER model, but the diagramatic conventions become quite complex and it is clear that the new model is intended to be used as a CASE tool [19].

3.1.2 Methods based on Other Classical Semantic Models

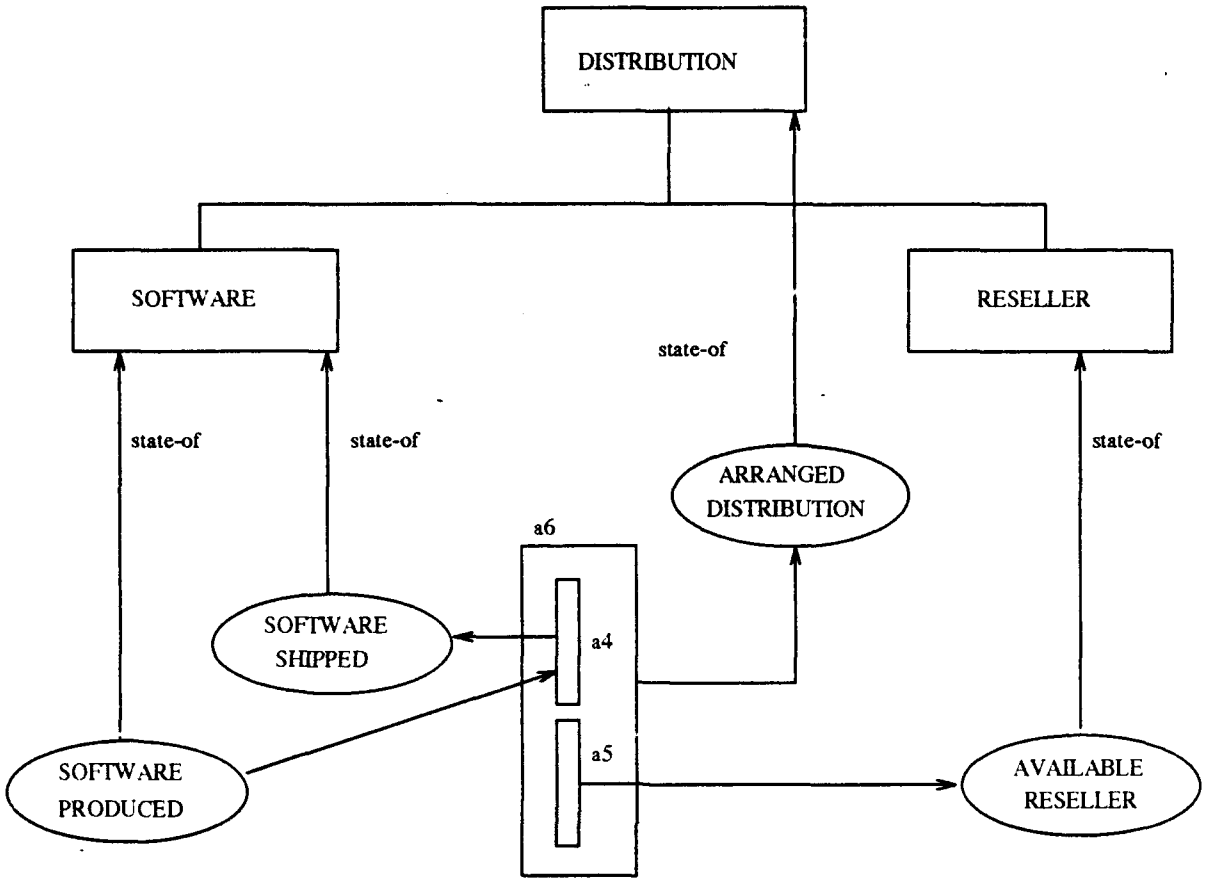
Rishe [23] presents the semantic binary model as an alternative to the E-R model. The binary model is more formal than the E-R model, but the expressiveness is similar. In contrast to the E-R model, the binary model deals with categories (classes) of concrete and abstract objects. Therefore, this model is structurally closer to the ODMS model, at least in terminology. The only relationships allowed are binary ones. Higher arity relations require additional objects to be formed, just as many ODMS's currently do.

The binary and E-R models are more alike than they are different. The diagrams which support the two models are similar. Moreover, like the E-R model, the binary model supports neither domain constraints nor any general procedural ability now common in ODMS's. However, it should be noted that Rishe's book includes some good criteria for database schema quality, which is the goal of database design.

A semantic data model which has had a greater impact on database technology is SDM. Hammer and McLeod originally introduced SDM in 1981 in response to the limitations, in their view, of the record-oriented database management systems (DBMSs) of the time. The purpose of SDM was to capture more of the high level semantics of the world than was possible by other models. Many of the ideas of SDM are now supported by ODMS's [16, 17].

Briefly, SDM deals with entities which may be organized into classes. Entities might be concrete objects, such as software applications or software resellers, or they might be events, such as shipping a software release. Entities can also be aggregations of other entities, for example, a software product, or even classes of entity identifiers, such as names for software products. Classes may be defined independently of other classes or comprised of collections of them. Database classes are generally related by interclass connections. In particular, subclasses and general groupings are supported. The idea of an object name (identifier) is explicitly supported. Entities and classes have attributes, which may consist of entities (possibly of a literal class), collections of entities, or values which are derived algorithmically. The semantics of attributes are quite specific in SDM. For example, many properties characterize an attribute: applicability, number of values, ability to change, ability to be null, relationship to other attributes, etc. Attributes may be related to other attributes as inverses or as matching attributes, which correspond to some of the ways SDM defines binary and n-ary relationships, respectively [16].

A key idea in SDM is *relativism*, which is essentially unbiased treatment of alternative views of the database. For example, an inverse relationship between attributes can be used to model, equally well, an entity comprised of two other entities. In either case, a binary relation is at the core [16].



a6: arrange-distribution (complex begin activity)
a4: ship-software (dependent end activity)
a5: accept-distribution (dependent inner activity)

Figure 4: Software Products Distribution (BIER Model)

SDM meets all structural requirements of a design methodology for OODB implementation purposes, but it may be too complex to serve as good communication bridge between users and developers. Furthermore, like the E-R model, it fails to adequately address the procedural abilities of an ODMS.[2] However, SDM was never intended to help design an OODB, predating the development of the first extensions into the object world by almost a decade. The contributions of SDM are more in terms of advocating a direction and destination for database systems. Having arrived, we can clearly appreciate the advice [16, 17].

Other semantic data models exist, including a new one called Generic Semantic Model (GSM) which appears in [17]. GSM is based largely on SDM, the E-R model, and the functional data model, but covers most features of other semantic data models. Like SDM, GSM is structurally adequate, but behaviorally lacking for OODB design.

3.2 Object-Oriented Models

Object-oriented programming, and languages to support it, arose because of a perceived need to ease the implementation and maintenance of computer models of real world objects and processes. The object-oriented paradigm focuses on active

[2] Derived data is only one sort of behavior associated with active data elements, not to mention more general procedures which come with a computationally complete environment.

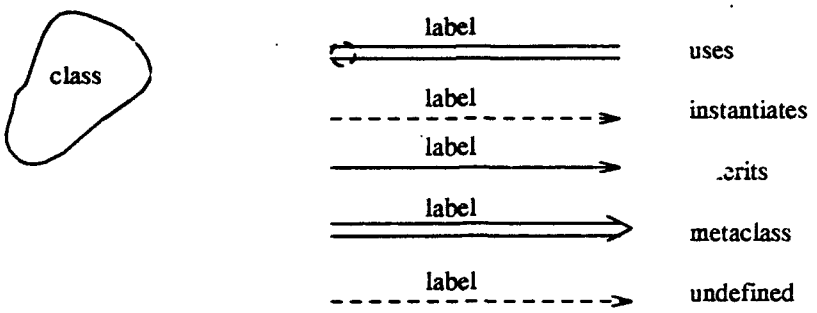


Figure 5: A class diagram (Booch)

data objects which encapsulate procedures with data. Active data objects provide for data and procedural abstraction enforced by well established interfaces. True object-oriented languages support a hierarchy of types, allowing some type of inheritance of variables, data values, and procedures from super-class to sub-class.

The newest database management systems have exploited the object-oriented paradigm in an effort to better model some applications. However, there are differences between requirements for database systems and those for general programming languages. For example, database systems are still mainly about data, not procedures.[3] Third generation database systems have been charged with retaining the benefits now enjoyed by relational database users and developers [27]. Therefore, we can expect that perhaps 90 percent of an OODB will depend on what we have been calling the structural elements.

3.2.1 Design Methods of Object-Oriented Programming

Despite differences between databases and general programming applications, design methods for object-oriented programming are are also valuable for OODB design [9]. Fortunately, methods for designing object-oriented programming applications are relatively advanced. The diagrams and methodology of Booch [4], Meyer [21], Shlaer and Mellor [25], and Rumbaugh et al. [24] are increasingly well-known and adopted.

Booch recommends using a variety of diagrams and notation for describing classes, objects, modules and processes. Booch's class diagrams include a rich notation for describing inter-class relationships and associated cardinalities. See Figure 5. A class template consisting of structured text accompanys the class diagram. Booch's object diagrams are similar to class diagrams, but focus on the message communication between classes. Object diagrams have special notation for timing and synchronization issues of message passing. We do not cover these issues as they are not important to OODB design because the ODMS manages them. Booch's well-known module diagrams show how classes and objects are allocated in the physical design of a system. They contain some of the same information as class and object diagrams. The difference is that module diagrams focus on the message interfaces between objects and the visibility of object data. See Figure 7. Booch also presents process diagrams which are not useful for OODB design [4].

Booch states that software design is an iterative process which requires broad and deep skills. He shuns the top-down approach, indicating that designers must work at multiple levels of abstraction simultaneously. Although he is against cookbook methodologies, Booch recommends a general design approach based on four steps [4].

1. Identify the classes and objects at a given level of abstraction.
2. Identify the semantics of these classes and objects.
3. Identify the relationships among these classes and objects.
4. Implement these classes and objects.

[3] Derived or virtual data (or relationships) is still viewed as data by the user. If database systems were not about data, how would they be differentiated from programming languages? Perhaps we are moving in this direction anyway, toward a blurring of distinctions, even about the differences between data and procedures.

Object name:	identifier
Documentation:	text
Class:	class name
Persistence:	[persistent, static, dynamic]
Processor:	[8088,80286,80386,80486,68000,68040]
Memory Amt:	integer
HD capacity:	integer
FD types:	[3.5,5.25]
Operating System:	text
OS Version:	text

Figure 6: An object template (Booch)

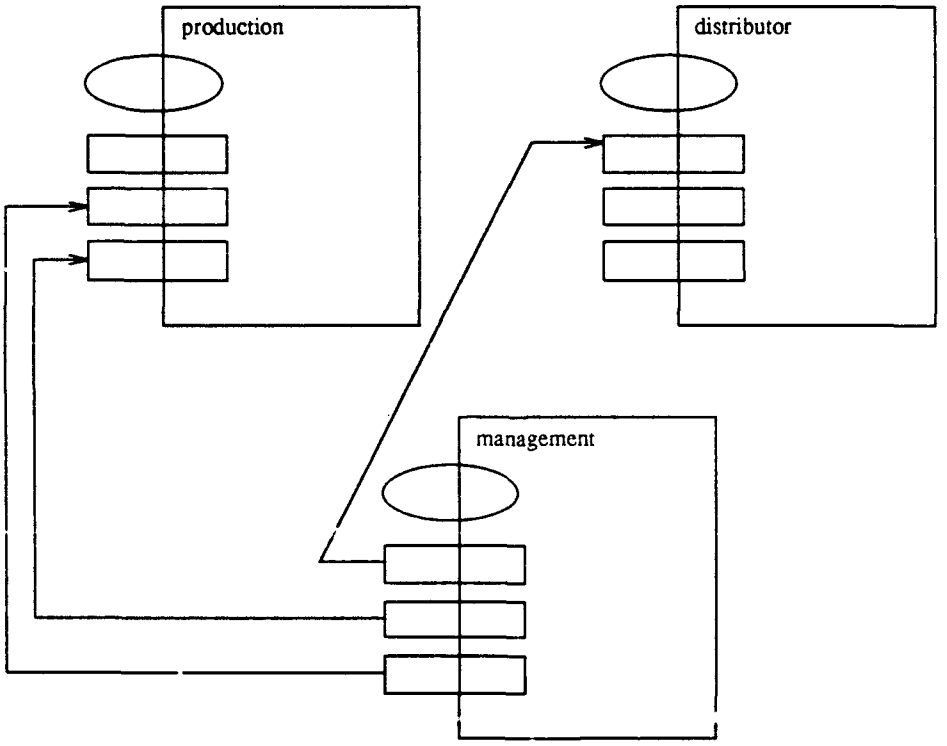


Figure 7: A module diagram (Booch)

For OODB design, this methodology seems reasonable, but it's rather abstract. It fails to stress the behavioral properties of objects. Furthermore, although Booch generally relates design activities to products, many of these products are of no value for OODB design. Some modification of this methodology is required for our purposes.

Recently, there have been noteworthy extensions to Booch's methods which increase their usefulness and semantics. For example, in [30], Wasserman, et al. propose a methodology and notation which essentially combines structure charts with Booch's notation, an approach previously investigated in [29]. The resulting notation, called object-oriented structured design (OOSD) notation, attempts to capture additional semantics to achieve the following goals: machine architecture independence, support for inheritance, reuse of parts of modules, language independence, independence of design approach,[4] and simplicity.

The basic expressiveness of OOSD is characteristic of another model based on the Objectchart notation introduced in [8]. Objectcharts are extensions of Harel's Statecharts, which are an application of a more general visual formalism, the higraph, also developed by Harel [12]. We'll delay discussion of the Objectchart method until we take an extended look at Harel's notation.

Another paper by some of the same authors describe how to formalize structural, functional, and dynamic models to avoid ambiguity, inconsistency and gratuitous detail [14]. The structural model normally used is the object model, a form of E-R model.[5] The authors describe the typical dynamic model as a state machine and improve upon it using Statecharts, a higraph representation of state diagrams. Similarly, the authors describe the typical functional model as a data flow diagram (DFD) which they reject in favor of specifying operations using pre- and post-conditions of operations. In so doing, the authors have arrived at a consistent set of design tools with which they develop a design methodology. The design process appears in the list below, which is taken directly from [14].

1. Develop a natural language description of problem.
2. Build an object model together with a data dictionary of classes, relationships, and attributes. Check completeness by stating consistency conditions between relations.
3. Derive an object structure model from the object model.
4. Use the object structure model to define a declarative functional model.
5. Use the object structure model to define a dynamic model together with diagrams showing examples of how events flow through the system in response to system operations.
6. State reasoned arguments and test case event flows to show that the dynamic model and functional model are consistent.
7. Iterate through the steps looking for missing classes, relationships, attributes, events, etc.

3.2.2 Object-Oriented Models for Database Design

Recently, researchers have begun to develop data models expressly for use in designing OODBs. While this work is still in its infancy, there are two articles which apply. Certainly, we can expect to see more published on OODB design, especially if ODMSs stabilize in terms of standard features and abilities.

In [28], the object-oriented semantic model for exception accommodation (OSEA) is developed. OSEA integrates formal semantics into the object-oriented data model while extending the object-oriented data model in terms of exceptions. Unfortunately, the resulting model is even more complex than the original object-oriented data model.

In [15], the structured entity model (SEM) is developed and used in an example of relational database design.[6] The goals of SEM are similar to the goals of our design method. The methodology for constructing a model based on SEM uses entity decomposition and coupling. These processes are not fully described in the paper. Decomposition apparently

[4] I.e., the notation should equally support a structured or behavioral bias.

[5] The authors state that the E-R model is based on formal semantics, unambiguous, and abstract. Still, they 'refine' it to an object structure model, which appears to be simply a class definition (template) for entities, attributes, and relationships.

[6] SEM itself is based on an object-orientation, yet the example is for a relational database. The authors indicate that SEM is not fully developed and state that the use in OODB design is a current research direction.

process until an entity appears twice on an SEM diagram. Once an entity appears twice, it can be coupled and the SEM diagram may be reduced. Diagramatic notations represent cardinality constraints and the arity of the relationships.

The authors present some results from empirically testing database designer performance, comparing SEM to the E-R model. These results suggest that SEM allows more accurate database designs, better recognition of complex relationships, and fewer unnecessary complex relationships. However, the authors conclude that more research is necessary to verify these findings. While SEM is too immature for our purposes, it has potential for contribution to OODB design.

3.2.3 Other Models

Although not directly applicable in the search for a OODB design method, there are two papers which are interesting in terms of the directions they suggest.

In [11], Hall and Gupta argue that the dynamics of a conceptual data model are best represented by the state changes they produce. Entity attribute values contain the effects of procedures, and the procedures themselves are not important. This is a functional view and reinforces the ideas presented in [14]. In addition, [11] presents the idea of entity dynamics into and out of different classes as a result of transitions which occur due to update.

In [16], a meta model for object-oriented data models is introduced. The idea of the meta model is to determine the capabilities and semantics of a number of different object-oriented data models for comparison purposes.

4 Higraphs and Statecharts

Earlier we discussed papers that used some form of Harel's Statecharts [8, 14]. In this section, we introduce higraphs and show how they may be used to augment the E-R model for OODB design. Higraphs can unify diagramatic notation and somewhat improve the E-R diagram with respect to representing generalization. We also show how Statecharts or Objectcharts may be used to model behavioral aspects of active data.

4.1 Higraph Semantics

Harel developed higraphs by combining the ideas of graphs, hypergraphs, and Venn diagrams (Euler circles). Figure 8 is a representative higraph which we will use to illustrate the accompanying semantics. Figure 8 is a collection of labeled rectangles and directed edges. The rectangles represents sets of things and, just as in Venn diagrams or Euler circles, a rectangle which is completely inside another rectangle represents set inclusion. For example, in Figure 8, the set S is a subset of set R. Unlike Venn diagrams, an overlapping of rectangles doesn't necessarily mean intersection. This is because higraphs require that all meaningful sets have their own rectangle. Therefore, overlapping rectangles without some rectangle in the interior of the overlapping area don't have any significance. Overlap represents intersection only if there is another rectangle inside [12].

A consequent of these semantics is that the *atomic* sets, those whose rectangles have no rectangles in their interior, are the real, identifiable sets. Rectangles with other rectangles in them (but no dotted lines) represent sets that are disjoint unions of atomic sets. For example, set A in Figure 8 is the union of elements in sets B and C, and there are no common elements in B and C. A dotted line which divides a rectangle is the higraph notation for unordered Cartesian product of set elements. A special labeling notation for dotted-line rectangles consists of putting the label in a tab attached to the rectangle, as in D and J. As an example of Cartesian product, the set J is comprised of W $\otimes$ X which means that, if there exist elements e1 $\in$ K $\cup$ N and e2 $\in$ I $\cup$ L $\cup$ M, then an element of J is the unordered tuple {e1,e2} [12].

The directed edges in Figure 8 are unlike those in more familiar graphs in that they connect sets to sets, rather than elements to elements. There are alternative interpretations of these edges. One meaning is that each element of connected sets are connected. Another is that *some* elements in each of the connected sets are connected [12].

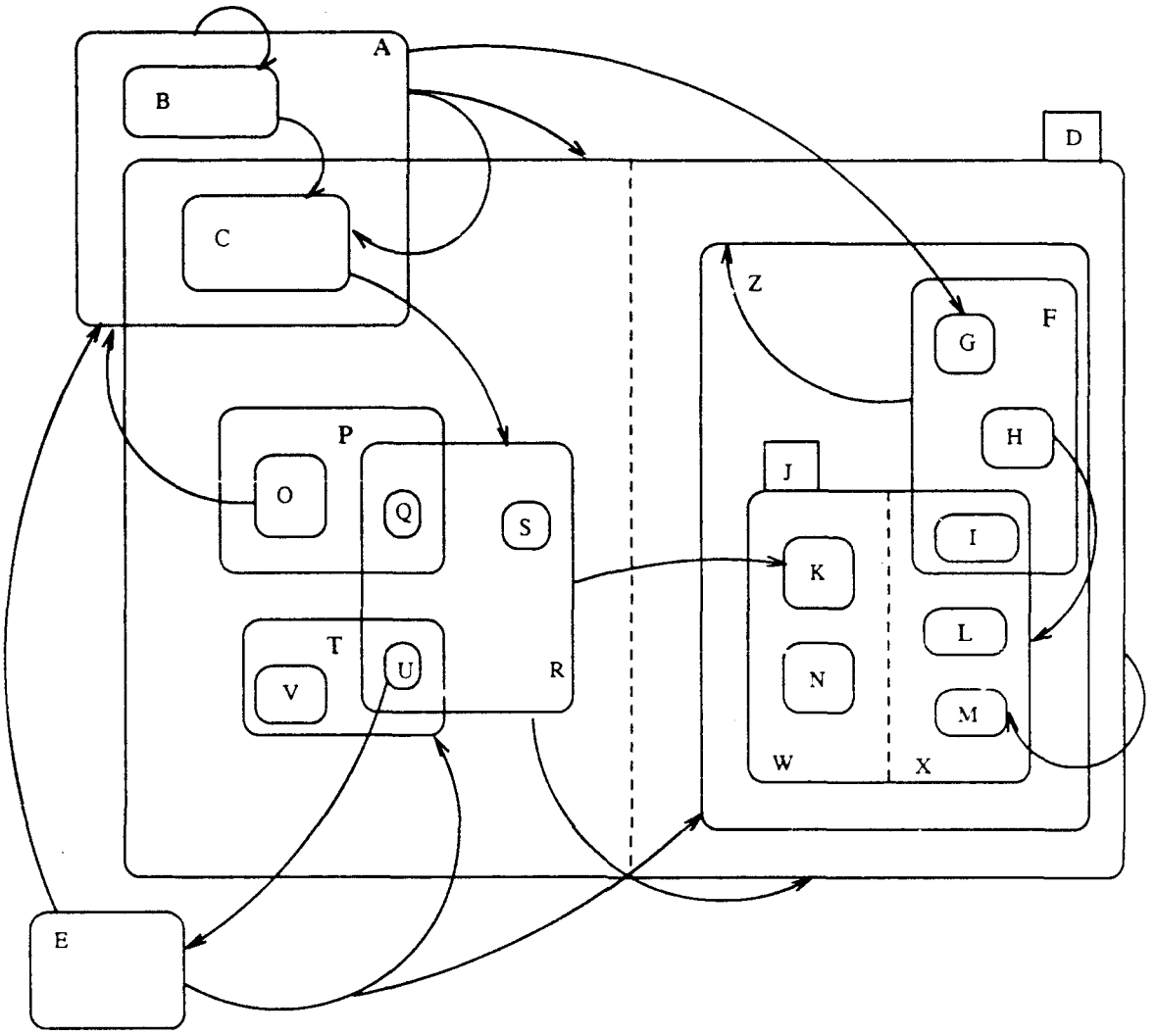


Figure 8: A higraph example

4.2 Higraphs as E-R Diagrams

There are a number of applications of higraphs. The most interesting from our standpoint is as an improvement to the entity-relationship diagram. For example, a higraph version of the entity-relationship diagram shown in Figure 1 is shown in Figure 9.[7]. Actually, Figure 9 represents all the information in Figure 1 and some additional information as well.

The most dramatic difference between Figure 1 and Figure 9 is that the IS-A relationships are now handled more cleanly. In Figure 1, IS-A (generalization) relationships are treated just like any other relationship. In the alternative notation of Figure 2, IS-A relationships gets its own special symbol. However, clearly this is also a flat representation, dependent only on shapes and links that connect them.

In Figure 9, topology is used to convey the IS-A relationships. Topology adds an extra dimensions to graphical notation and thus conveys more information than using a flat notation. For OODBs, which are normally hierarchical in structure, a higraph representation will scale better than the flat E-R diagram. To illustrate this, see Figures 10 and 11. These diagrams represent the same information, but the higraph uses four fewer symbols and no links.

[7] The idea of exploiting topology for semantic networks and models is not particularly new, but has not taken hold for some reason [12]

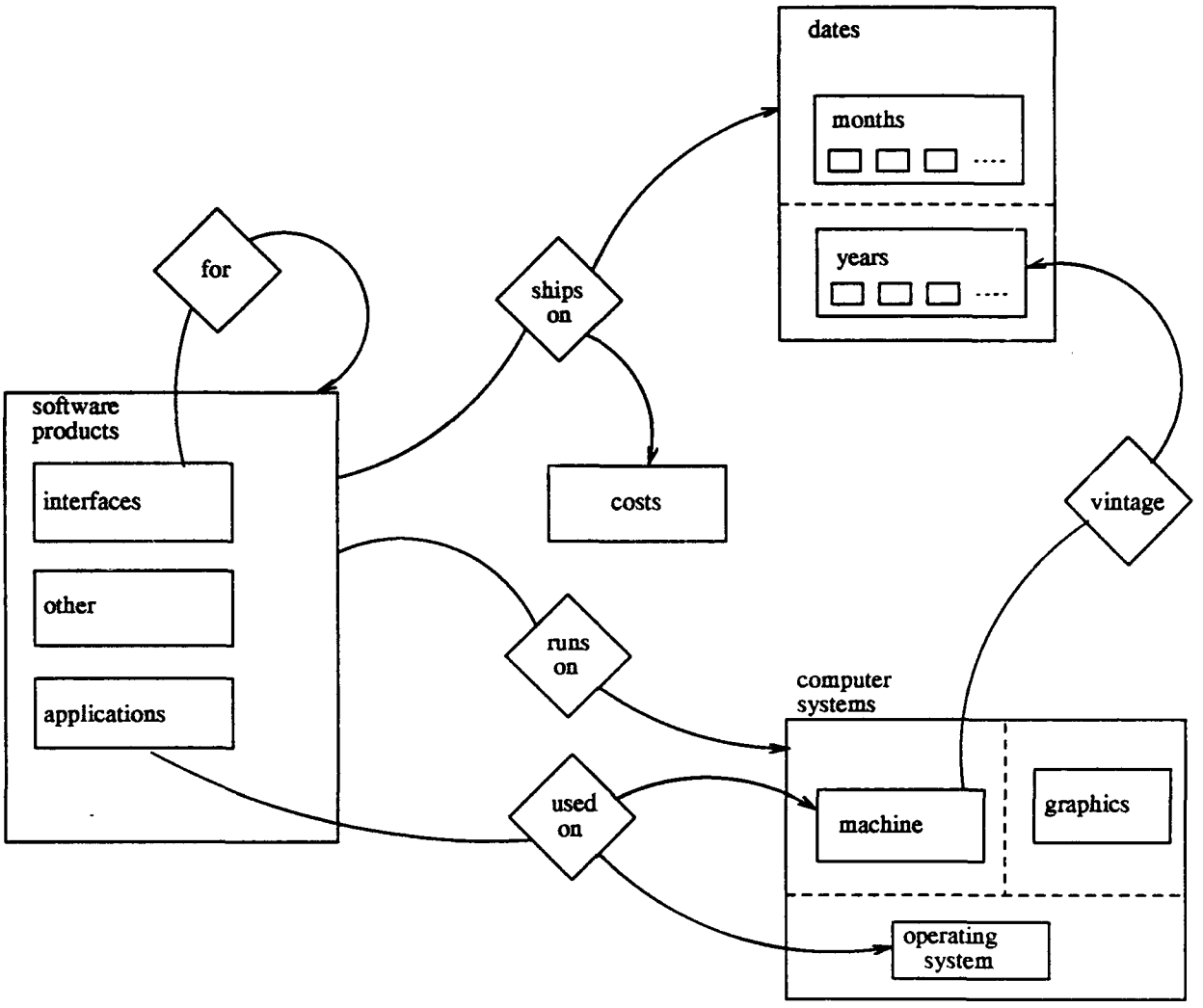


Figure 9: A higraph version of an entity-relationship diagram

When the overlaying of rectangles becomes too tangled or rectangles become too small, internal rectangles can be blown up in another diagram to show more detail. For example, the *computer systems* rectangle in Figure 10 is represented in more detail in Figure 9, which indicates that computer systems are Cartesian products of a machine, an operating system, and a graphics capability.

The differences between the two notations become more pronounced as we add requirements to our design. What if we want to access particular sets of software products based on the programming language that they are implemented in? Figure 12 shows how we can do this using a higraph E-R diagram. In that diagram, we demonstrate that we can have relationships involving Lisp interfaces and Ada applications. We also represent a subroutine calling relationship between Lisp programs and the Ada routines. The diagram currently has only a calling relationship between Lisp interfaces and Ada applications, but adding the set of, say, Lisp applications requires only one more rectangle.

Using the normal E-R diagram, representing these additional entity sets adds many more symbols and forces choices as to how the diagram is to be structured. In Figure 13, we have chosen to make the first distinction on product type, while in Figure 14, we have classified first on implementation language.

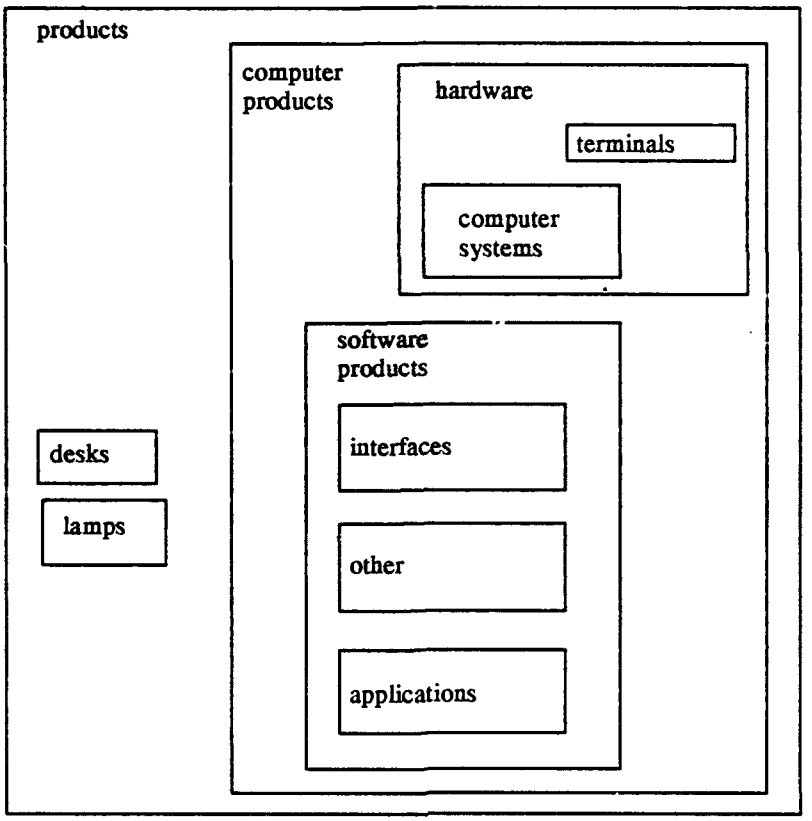


Figure 10: A hierarchy of sets

Figure 9 also shows the use of the Cartesian product for the partitioning of dates. The SHIPS-ON relationship between software products, costs, and dates requires both month and a year information, but the VINTAGE relationship is only between machines and years, because less detail is required. Representing this situation using the current E-R diagram requires making an unnecessary design choice. One alternative is to make months and years separate entities. Then, SHIPS-ON becomes a four-way relationship. Another alternative is to have a month-and-year entity and also a year entity. In this case, the design seems redundant.

Notice that attributes do not appear in Figure 9. These can be added as itemized lists inside rectangles, for instance. Attributes of relationships can also be added, but are not as important for designing OODBs because relationships are represented as attributes in ODMS applications. Specific cardinalities could also be added.

4.3 Statecharts as formal state machines

Another application of higraphs is Statecharts. Statecharts model transitions between states like state diagrams do, but have two other useful features. One is that orthogonal activities may be modeled, and the other is that communication between orthogonal activities is supported. Let's review the semantics of Statecharts.

To support the discussion, consider Figure 15. The diagram has a large rectangle, representing state Y, that is partitioned between states A and D. States A and D are not atomic, but are *exclusive-or* generalizations. That is, being in state A means the same thing as being in either state B or state C, but not both. Just as in state diagrams, the arrows represent transitions which correspond to the labels on the arrows. A transition may have a precondition, shown in brackets, attached to it. One such transition in Figure 15 is f which fires only if [in(G)] is true. An initial state is denoted by an unconnected arrow pointing to it. The states B and F are initial states in Figure 15.

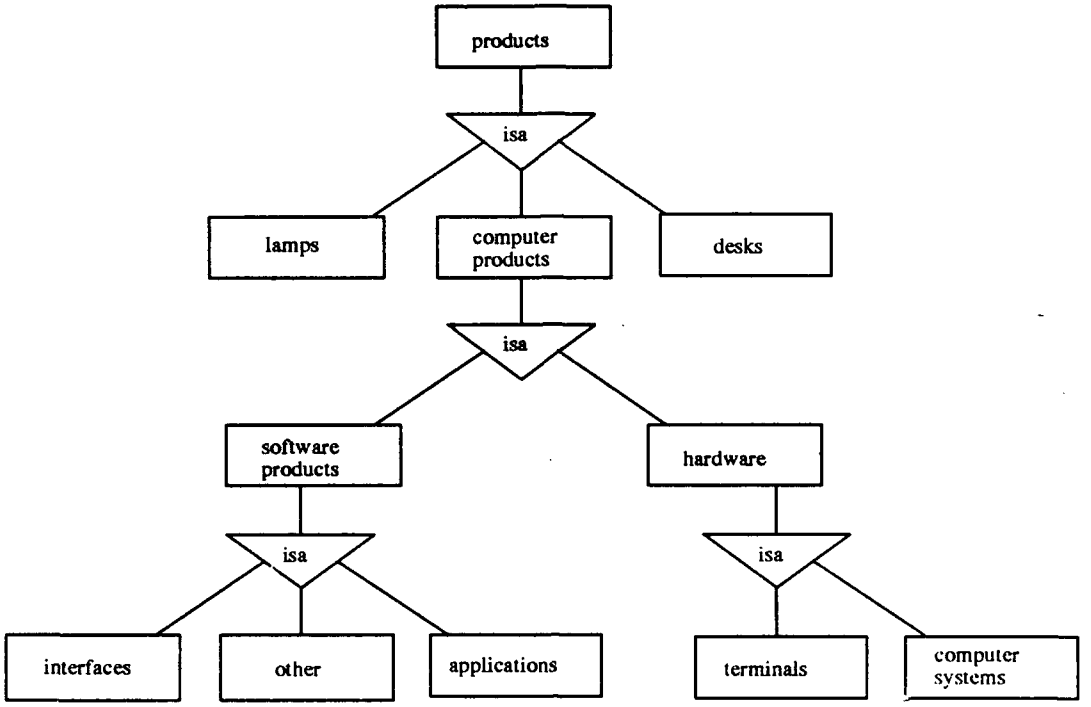


Figure 11: An E-R representation of hierarchy

The partition of state Y between general states A and D represents *orthogonality*. In other words, being in state Y is tantamount to being in states A and D at the same time. For example, initially, the system is in states B and F. Orthogonality is the *and* complement of generalized states; *and* is a natural dual of *exclusive-or*.

Let's step through some transitions. If the k transition occurs from the initial state, the system is in states B and E. Each time state change occurs, the state change is broadcast globally. If the e transition occurs, the system is afterwards described by being in states C and E. At this point, an f transition will change no states, but if a g transition occurred just previously, then [in(g)] would have been true, and f would then cause a transition from state C to state B.

Notice that, just as in the case of higraph-based E-R diagrams, Statecharts take advantage of topology to reduce complexity over the flat state diagram. For example, the transition p causes a change from any state-combination in Y to the I state. This would require separate transition arrows from each state in a flat diagram. Furthermore, orthogonality reduces the number of state representations required. In a flat diagram, separate states are required for each *combination* of states possible, i.e., BE, BF, BG, ... six versus five for Figure 15, but Figure 15 isn't really representative. This is a combinatorial issue. Clearly, Statecharts require the sum of the states in their partitions, while a flat state diagram would require the *product* of the states.

Two other characteristics differentiate Statecharts from state diagrams. First, the *exclusive-or* decomposition of states supports top-level design and allows the use of abstraction. Perhaps it isn't important whether the system is in state B or C, or even if it is in A and D. Maybe you just need to know whether the system is in Y, H, or I.

Second, Statecharts are inherently parallel, while flat state diagrams are necessarily sequential. Orthogonal components change states simultaneously, so Statecharts would seem not to be biased toward the Von Neumann model of computing.

Statecharts are useful in the specification and design of almost any kind of reactive system.[8] Harel, in [12], uses Statecharts to describe the workings of a digital wrist watch. See Figures 16 and 17. In his example, the watch buttons, which are labeled a, b, c, and d, cause state transitions which are changes in watch displays or time variables. Harel

[8] Databases are just such event-driven, reactive systems.

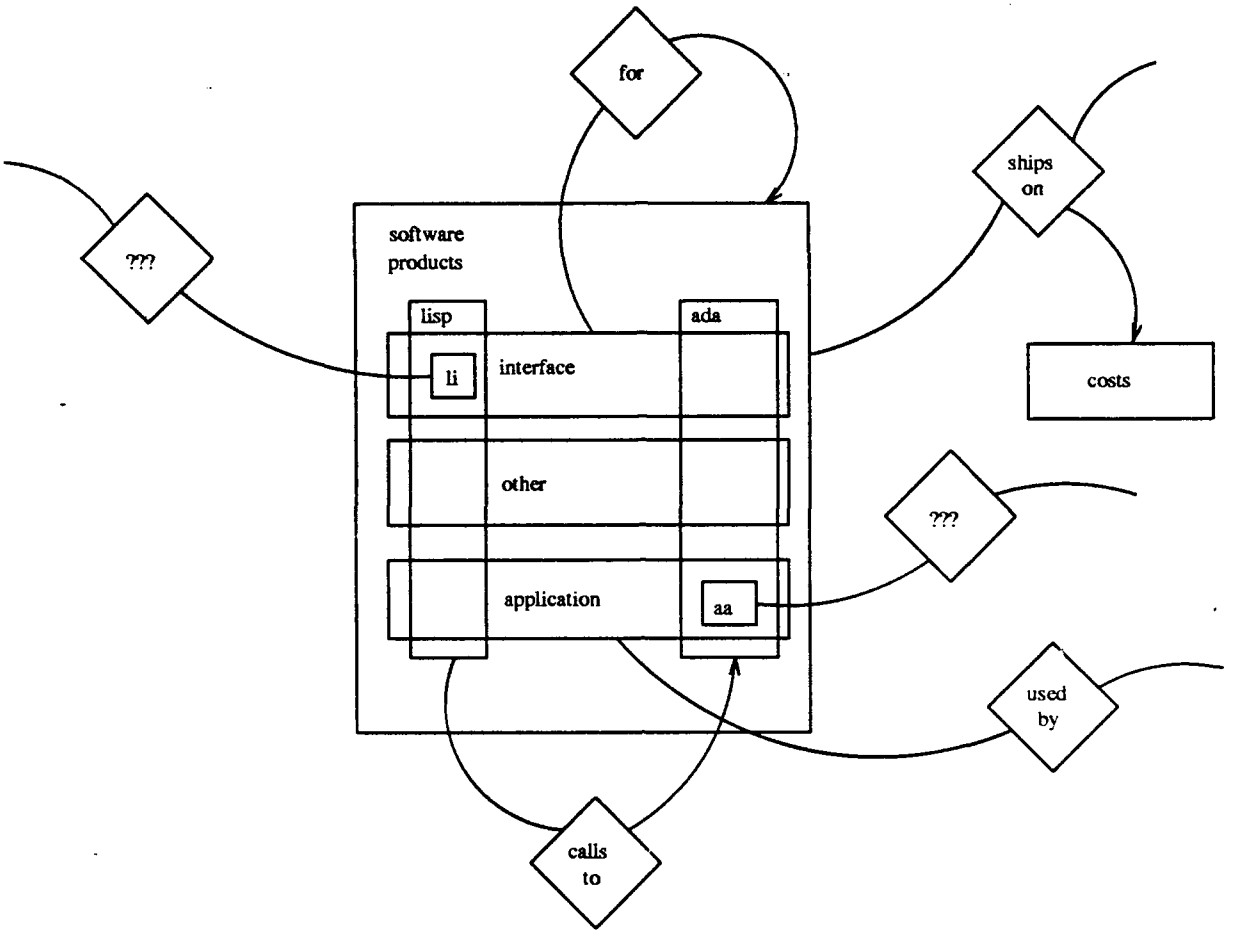


Figure 12: Another slice of software products

shows that pressing button b, which orthogonally controls the light on the watch, has other effects such as returning from update display states and starting the stop watch. Why doesn't button b activate the light only when the watch is in the time display state? That would seem to be a better design.

4.4 From Statecharts to Objectcharts

We have discussed higraphs and Statecharts thoroughly because they have some very nice characteristics, and more research is needed to find ways of exploiting them as OODB design tools. While some researchers have already begun to use them, a firm basis in Harel's *original* ideas will allow broader appreciation and perhaps some new ideas. The interested reader is invited to examine [12] closely. Now it's time to return to the design method based on the Objectcharts of [8].

In [8], it is suggested that conventional E-R diagrams be used to build an information model of the application domain and that class behavior be specified using Objectcharts in a second modeling step. After this introductory suggestion, the paper focuses solely on the use of Objectcharts to model dynamics.

The behavioral properties of objects depend somewhat on structural architecture, which shows the *service* (method) interface between objects. In [8], configuration diagrams like Figure 18 are used to describe the service interfaces between objects. In the notation, solid lines represent provided services and dash lines represent requests for service.

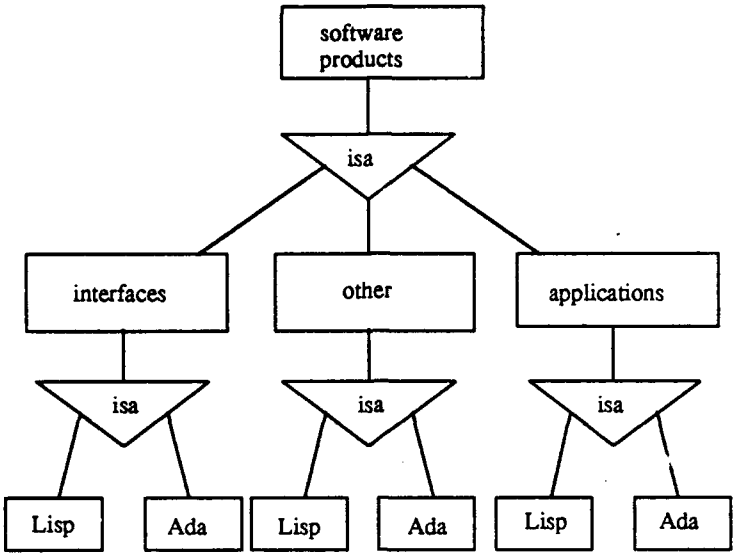


Figure 13: A breakdown using product type

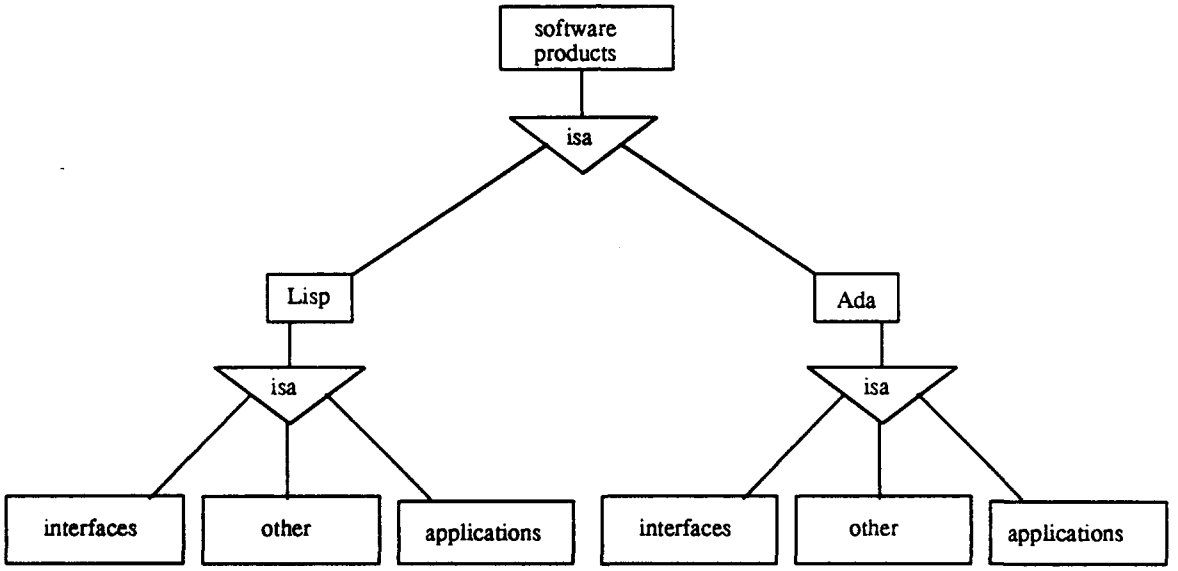


Figure 14: A breakdown using implementation language

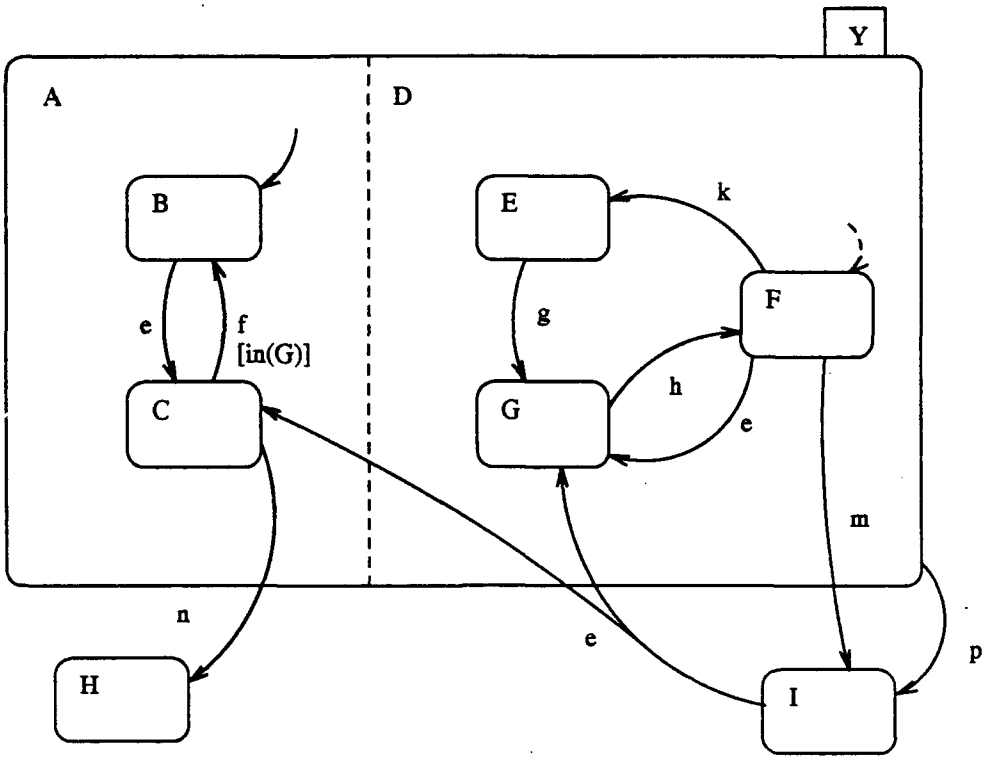


Figure 15: A Statechart

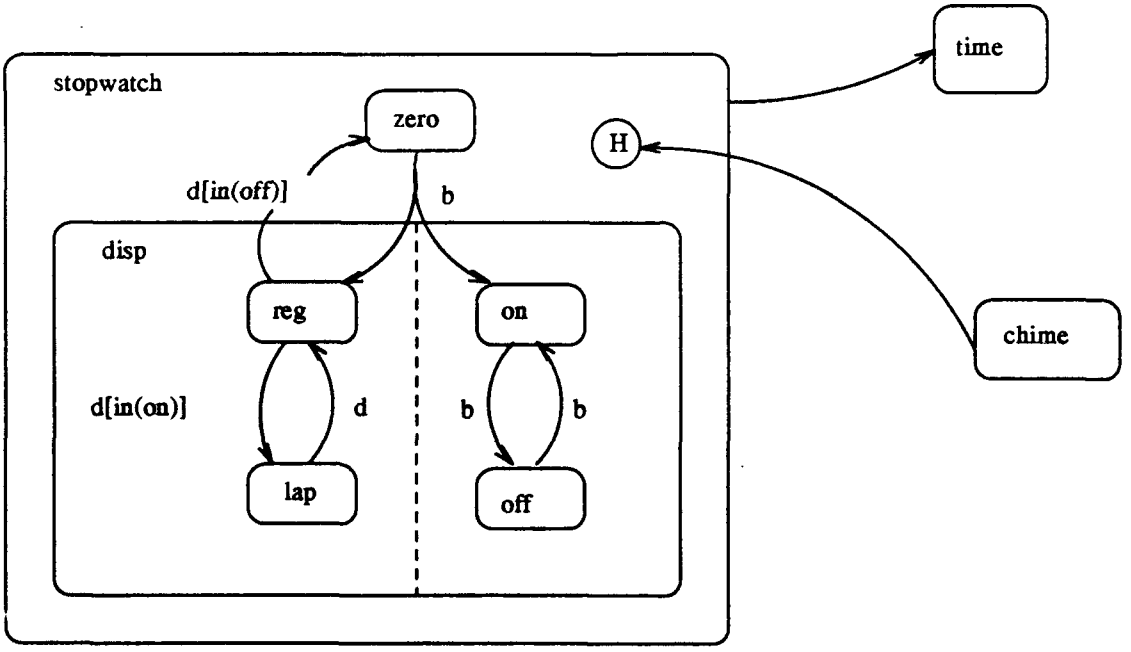


Figure 16: The stopwatch details

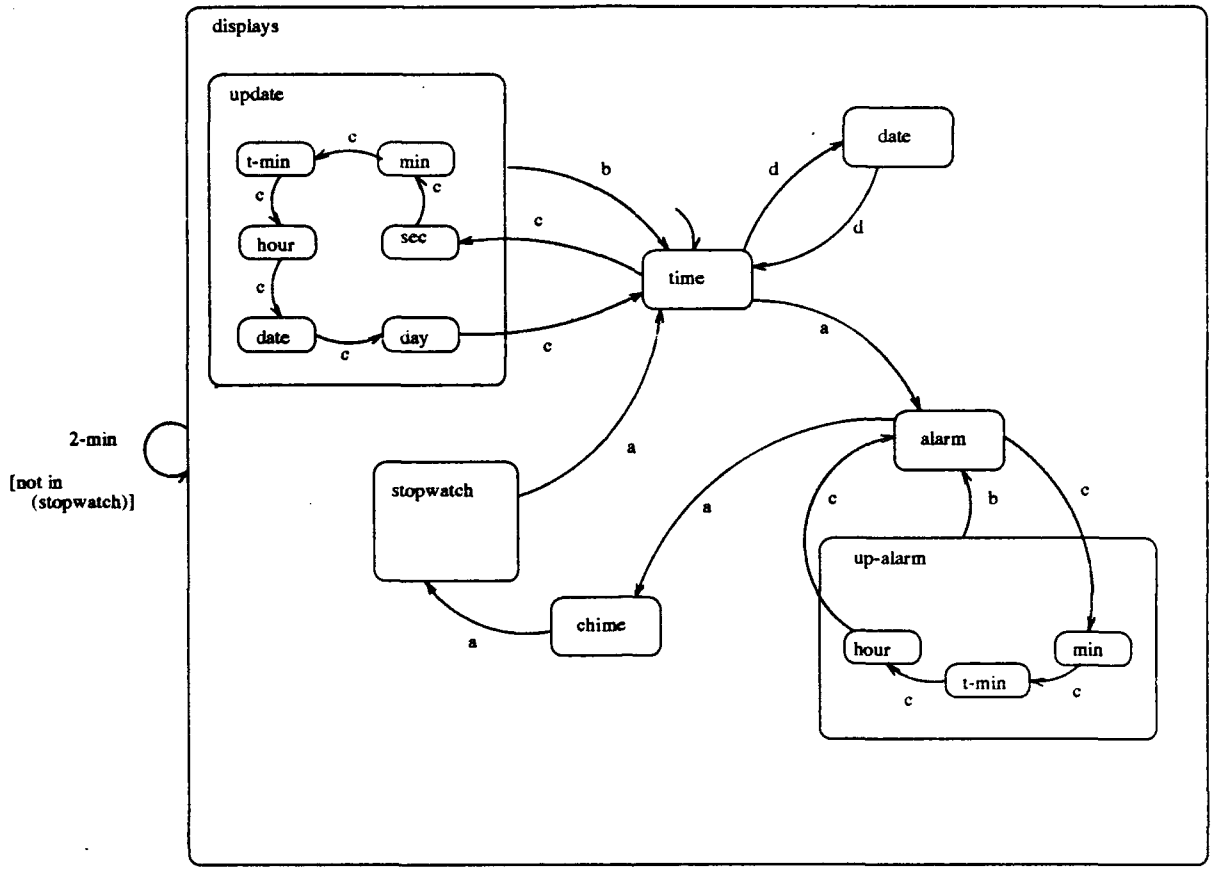


Figure 17: A digital watch

Configuration diagrams are similar to diagrams of Shlaer and Mellor's Object Communication Model [25] or Booch's object diagrams [4].

Figure 18 is the configuration diagram of an alarm clock on a graphics workstation. The alarm clock class requires *open window* and *close window* services from the window class and a *time?* service from a clock, which is of system clock class. The alarm clock class provides services such as *set, cancel, stop, time of day,* and *alarm time.*

The configuration diagram does not specify procedural control or results of state transitions on objects. Objectcharts do this. Figure 19 shows an Objectchart of an alarm clock in a windowing interface on a graphics workstation. Objectcharts extend Statecharts by including the effects of transitions on attribute values. For example, in Figure 19, the attributes *finish, alarmtime,* and *timeofday* are represented. Types are specified on the diagram as well; in this case, all variables are of type *time.*

The behavior of the alarm clock is represented in the Objectchart shown in Figure 19. Initially, the alarm clock is in the *alarmoff* and *timeupdate* states. The *timeupdate* state undergoes a transition every second which has the effect of updating the *timeofday* variable, that is the alarm clock is accurate to within 1 second or so. Setting the alarm clock causes a state change to the *alarmon* general state, with initial specific state *quiet.* A service request to the bell (window class) causes the clock to 'ring,' meaning a window becomes active on the workstation screen. If an external object requests a *stop* service from the alarm clock, or if a certain amount of time elapses, the ringing halts, and the alarm clock re-enters the *quiet* state, from which a *cancel* transition can return the alarm clock to the *alarmoff* state.

Transactions in the object model arise when an object requests or provides a service. The *life-cycle* of an object has become a popular term meaning the various states possible and the transitions between them. Therefore, the life-cycle

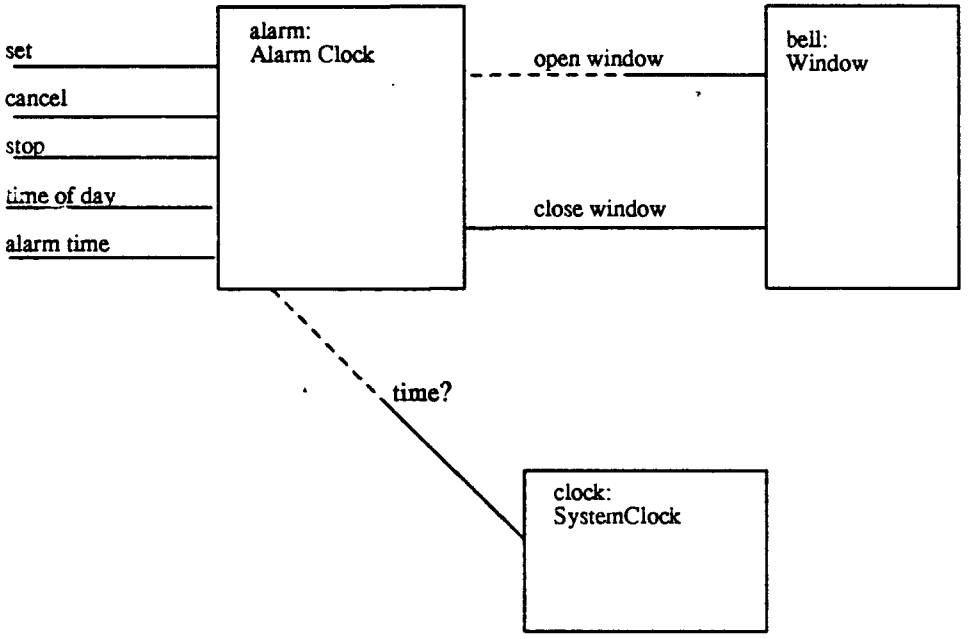


Figure 18: An Alarm Clock Configuration Diagram

is entirely dependent on services requested or provided. Provided services are depicted using simple labels like *set* and *cancel*. Requested services include an address indicator, as in */W.openwindow*. Objectcharts deviate from statecharts by disallowing broadcast communication, which the authors felt was an incorrect model for objects, hence the requirement for addresses. The configuration diagram is used to decode the address indicator, if necessary. Not all services cause transitions. In [8], these are termed *observers*.

An Objectchart does not fully describe transitions. Transition pre-conditions, post-conditions, and invariant conditions are specified in an adjoining writeup.

Contrasted with Harel's presentation of Statecharts, Objectcharts seem to be overly complicated. Figure 19's depiction of transition is dependent on a particular object configuration, which may be too detailed for conceptual design. Furthermore, it seems to us that Figure 19 should include, for example, a precondition on the transition *W.closewindow* which is related to *finish*. Therefore, Objectcharts are not entirely satisfying as a design tool. However, some form of Statechart seems to be the best descriptor of object behavior currently suggested in the literature. It's easy to show that Statecharts, which are basically state machines, are at least as expressive as Petri nets or other dynamics notations in popular use. From this expressiveness, we can extrapolate that *any* arbitrary type of procedural or behavioral property can be modeled with Statecharts.

5 An Eclectic Conceptual Model for OODB Design

None of the methodologies suggested in the literature seem to meet all needs of OODB design, although the methodology in [14] seems ample for more general object-oriented software design. We can blend this methodology with the best parts of others to develop an *eclectic* design method.

We said above that OODBs are still about data, perhaps to a great extent. The E-R model is the clear choice for modeling structural aspects of an OODB, especially if we use a higraph-based E-R diagram. By using the higraph, we unify at least the diagramatic notation of our design method. When more detail is required, the E-R model can be supplemented with class/object definitions using structured text (templates), as in Booch's notation.

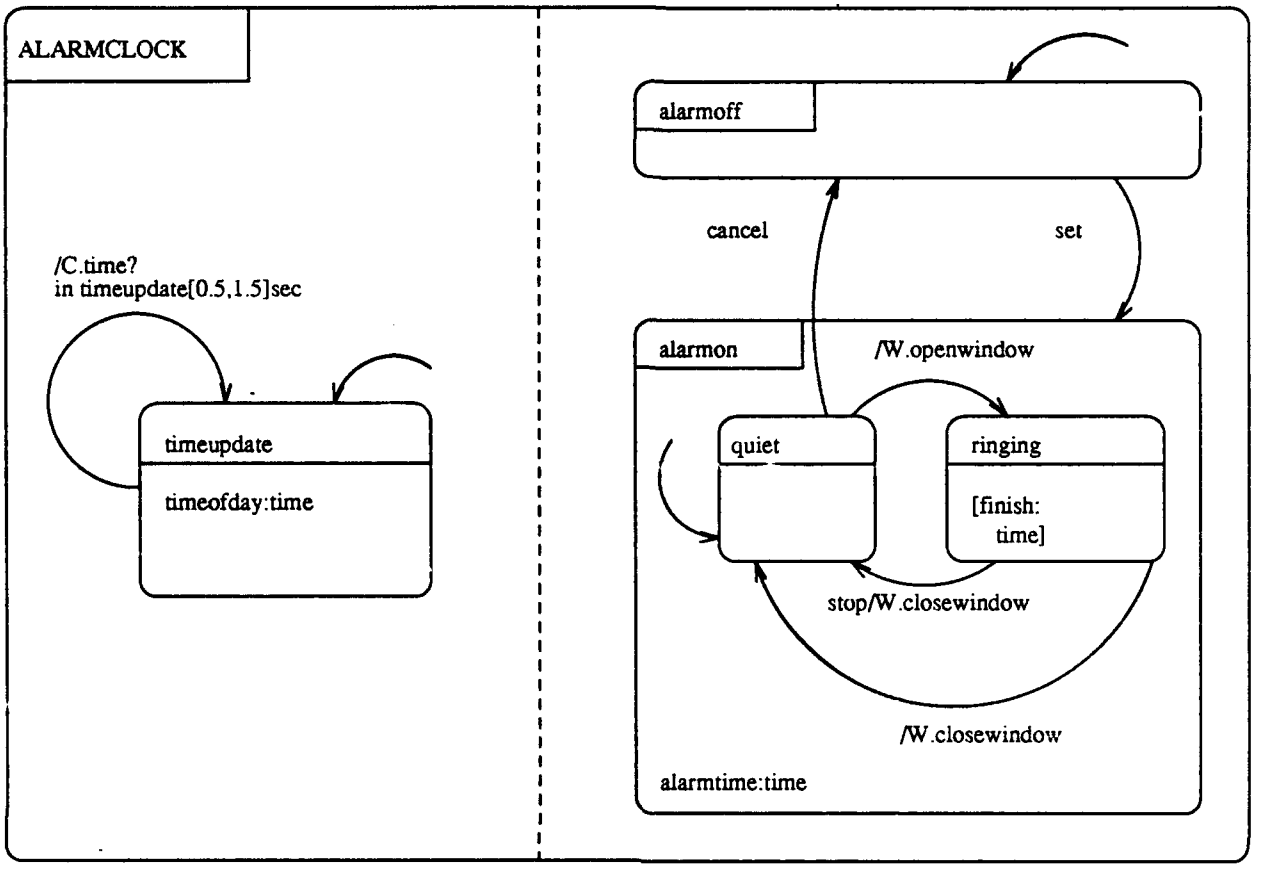


Figure 19: An Alarm Clock Objectchart

To specify object behavior, we need a functional model, and to design active objects, we need a dynamic model which can be shown to be consistent with the functional specification. We should take as our functional model, the transition rule specification in [8, 14]. That is, for each transition, we must write down pre- and post-conditions, as well as invariants in a structured text form. Objectcharts, or some other modification of Statecharts, provide the best dynamic model and also can be used easily to show consistency with our chosen functional model. These choices support the argument extended in [11] that state changes are the best model of dynamics. Furthermore, Statecharts seem to be less complicated than the Petri nets of [18, 19].

Using these tools, we can adapt the methodology of [14] for OODB design:

1. Develop a higraph-based E-R diagram, supplemented by a natural language description describing constraints, rules, and other behaviors.

2. Use the E-R diagram and text description as a specification to develop a structural design for implementing the database. The design product should be a configuration diagram together with class and object templates. Check for completeness by stating consistency conditions between relations.

3. From the configuration diagram and class/object templates define a declarative functional model using templates of transition pre-conditions, post-conditions and invariants.

4. Use the configuration diagram and functional model to define a dynamic model using Statecharts or Objectcharts.

5. State reasoned arguments and test case event flows to show that the dynamic model and functional model are consistent.

6. Iterate through the steps looking for missing classes, relationships, attributes, constraints, events, etc.

How does this methodology compare with our stated needs? Obviously, it is much more complex than the E-R model we use for relational database design. As a result of the added complexity, it is doubtful that the model above will communicate with the same clarity and conciseness as the E-R model. However, the object-oriented data model is also more complex than the relational model, so there seems to be no way of avoiding some added complexity in any complete design method.

On the other hand, the object-oriented data model is more expressive than the relational model. Therefore, we can model some problems much more naturally and easily. This helps offset some of the complexity of the design. Some applications will be less complex to model, overall.

Furthermore, if it is true that the bulk of the database depends on relational-type activities, i.e., structural, rather than procedural aspects, then most of the design will be based on the E-R model. Therefore, the methodology above will retain as much simplicity as possible.

In any case, it is clear that the higraph formalism helps unify, to some extent, behavioral and structural properties.[9] Furthermore, by virtue of the additional depth provided by topology, higraphs have potential for simplifying diagrams, especially for complicated cases.

How does the eclectic methodology stack up against the goals we established at the beginning of the paper? One requirement was for iterative design, with a series of activities and products. The steps above outline these activities and products clearly. Another goal was for rich, yet simple design products which capture and communicate the design effectively. We have made an effort to provide these using the higraph notation. Our goal for the design process was essentially the same as Booch's methodology, which is included in the eclectic method we developed.

6 Conclusions and Future Work

Clearly, more research is required to determine the extent to which the conceptual model presented meets the needs of OODB design. The E-R and functional parts will not present a problem, but the use of Objectcharts might. Probably the first step is to model several typical ODMS applications as a sanity check. If the behaviors of simple applications can be modeled, then a rigorous search through the space of procedural possibilities should be undertaken. Can Objectcharts model constraints, methods, and agents adequately? If so, then alternative object communication semantics should be explored. Objectcharts currently limit information to that known by objects requesting and supplying services, although clearly other information may be required. If so, additional service invocations are needed.

Configuration diagrams and E-R diagrams might contain redundant information. Further research is required to determine if this is true and, if so, how the situation can be improved.

References

[1] Alagić, Suad. *Object-Oriented Database Programming*. Springer-Verlag, 1989.

[2] Bic, Lubomir and Jonathan P. Gilbert. "Learning from AI: New Trends in Database Technology," *Computer* (March 1986).

[3] Bobrow, Daniel G., et al. "CommonLoops: Merging Lisp and Object-Oriented Programming." *Research Foundations in Object-Oriented and Semantic Database Systems* edited by Alfonso F. Cárdenas and Dennis McLeod, Prentice Hall, 1990.

[4] Booch, Grady. *Object Oriented Design with Applications*. Benjamin/Cummings, 1991.

[9] The symbols are the same, the semantics are a little different.

[5] Cárdenas, Alfonso F. and Dennis McLeod. "Database Description with SDM: A semantic database model." *Research Foundations in Object-Oriented and Semantic Database Systems* edited by Alfonso F. Cárdenas and Dennis McLeod, Prentice Hall, 1990.

[6] Cattell, R. G. G. *Object Data Management: Object-oriented and Extended Relational Database Systems*. Addison-Wesley, 1991.

[7] Chen, P. P. "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems*, *1*(1) (1976).

[8] Coleman, Derek, et al. "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," *IEEE Transactions on Software Engineering*, *18*(1) (1992).

[9] Dyer, Douglas E. and Mark A. Roth. *Object-Oriented Design Unifies Databases and Applications*. Technical Report AFIT/EN-TR-92-2, Wright-Patterson AFB OH: Air Force Institute of Technology, July 1992.

[10] Gogolla, Martin and Uwe Hohenstein. "Towards a Semantic View of an Extended Entity-Relationship Model," *ACM Transactions on Database Systems*, *16*(3) (1991).

[11] Hall, Gary and Ranabir Gupta. "Modeling Transition." *Proceedings of the Seventh Annual Conference on Data Engineering*. 1991.

[12] Harel, David. *On Visual Formalisms*. Technical Report, Carnegie-Mellon University, 1987.

[13] Harel, David. "Statecharts: a visual formalism for complex systems," *The Science of Computer Programming*, *8* (1987).

[14] Hayes, Fiona and Derek Coleman. "Coherent Models for Object-Oriented Analysis." *ACM OOPSLA '91 Conference Proceedings*. 1991.

[15] Higa, Kunihiko and Olivia R. Liu Sheng. "An Object-Oriented Methodology for End-user Logical Database Design: The Structured Entity Model Approach." *Proceedings of the Thirteenth Annual International Computer Software and Applications Conference*. 1989.

[16] Hong, Shuguang and Fred Maryanski. "Using a Meta Model to Represent Object-Oriented Data Models." *Proceedings of the Sixth Annual Conference on Data Engineering*. 1990.

[17] Hull, Richard and Roger King. "A Tutorial on Semantic Database Modeling." *Research Foundations in Object-Oriented and Semantic Database Systems* edited by Alfonso F. Cárdenas and Dennis McLeod, Prentice Hall, 1990.

[18] Kappel, Gerti and Michael Schreff. "A Behavior Integrated Entity-Relationship Approach for the Design of Object-Oriented Databases." *Entity-Relationship Approach* edited by C. Batini, Elsevier Science Publishers B.V. (North-Holland), 1989.

[19] Kappel, Gerti and Michael Schreff. "Object/Behavior Diagrams." *Proceedings of 7th International Conference on Data Engineering*. April 1991.

[20] Korth, Henry F. and Abraham Silberschatz. *Database System Concepts* (2nd ed. Edition). McGraw-Hill, 1991.

[21] Meyer, Bertrand. *Object-Oriented Software Construction*. Prentice Hall, 1988.

[22] Navathe, Shamkant B. and Mohan K. Pillalamarri. "OOER: Toward Making the E-R Approach Object-Oriented." *Entity-Relationship Approach* edited by C. Batini, Elsevier Science Publishers B.V. (North-Holland), 1989.

[23] Rishe, Naphtali. *Database Design Fundamentals*. Prentice Hall, 1988.

[24] Rumbaugh, James, et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1988.

[25] Shlaer, Sally and Stephen J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data*. Yourdon Press, 1988.

[26] Smith, J. M. and D. C. P. Smith. "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, 2(2) (1977).

[27] Stonebraker, M., et al. "Third-Generation Data Base System Manifesto," *SIGMOD Record*, *19*(3):31–44 (September 1990).

[28] Tan, P. L. and T. S. Dillon. "The Conceptual Design of OSEA: An Object-oriented Semantic Data Model." *Proceedings of the Fourteenth Annual International Computer Software and Applications Conference*. 1990.

[29] Ward, Paul T. "How to Integrate Object Orientation with Structured Analysis and Design," *IEEE Software* (March 1989).

[30] Wasserman, Anthony I., et al. "The Object-Oriented Structured Design Notation for Software Design Representation," *Computer* (March 1990).

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	9 September 1992	Technical Report

4. TITLE AND SUBTITLE
An Eclectic Method for Object-Oriented Database Design

5. FUNDING NUMBERS

6. AUTHOR(S)
Douglas E. Dyer, Capt, USAF
Mark A. Roth, Maj, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/EN-TR-92-4

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
ASC/RWWW
Wright-Patterson AFB OH, 45433-6503

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

No dominant methodology has emerged for designing object-oriented databases. In this paper, we identify characteristics for a good design methodology and review the literature of supporting models and approaches. Semantic data models and object-oriented methods are presented. We then take an extended look at Harel's higraph notation and use it to extend the entity-relationship model for object-oriented database design.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Object-Oriented Design, Database Design, Software Systems Modeling	23
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL